NETGIS
Gesellschaft für
Geoinformation und Umweltplanung

Tutorial
„Einführung in PostgreSQL mit PostGIS unter MS Windows"

NETGIS Gesellschaft für Geoinformation und Umweltplanung, Trier © 2009
Dieses Dokument ist urheberrechtlich geschützt und wird unter der GNU Free Documentation License freigegeben (http://www.gnu.org/licenses/fdl.txt).

Empfehlung von weiterführender Literatur:

- PostgreSQL Administration, Peter Eisentraut & Bernd Helmle, O'Reilly Verlag 2008
- Beginning Databases with PostgreSQL, Neil Matthew & Richard Stones, Apress 2005
- WebMapping mit Open Source-GIS-Tools, Tyler Mitchel & Astrid Emde & Arnulf Christl, O'Reilly Verlag 2008

Online Dokumentationen:
Postgres: http://www.postgresql.org/docs/
PostGIS: http://postgis.refractions.net/documentation/

Vorbemerkung:
Dieses Tutorial wurde unter Verwendung der Versionen von PostgreSQL 8.3 mit der räumlichen Erweiterung Postgis 1.3.3 erstellt und getestet.
Die Installation wurde unter dem Betriebssystem MS Windows XP Prof. und MS Windows 2003 getestet.

1. Installation von Postgresql Version 8.3.x und Postgis auf Windows 2003 Server oder Windwos XP

Die aktuellen Windows-Versionen (Installer und Binary-Pakete) für PostgreSQL stehen hier zum Download bereit: http://www.postgresql.org/download/windows

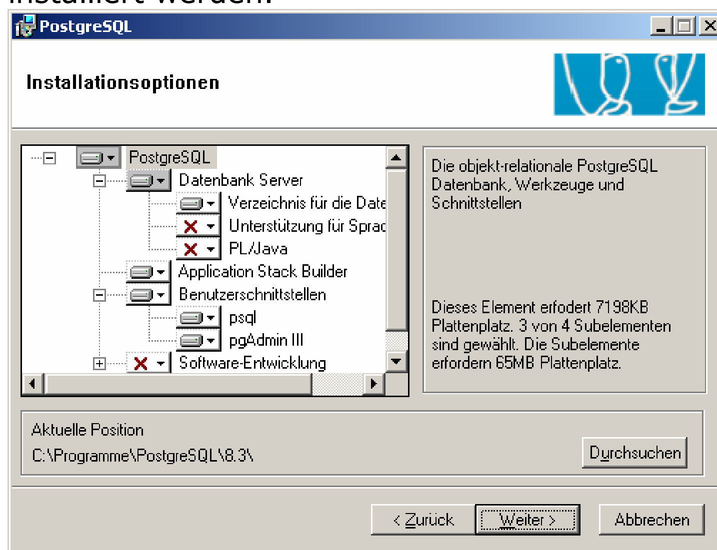
a) Installation PostgreSQL

1. Benutzer („**postgres**") ohne Adminrechte anlegen, Passwort für Benutzer vergeben: **xxxx** *(Start->administrative Tools–>Computer Management)*

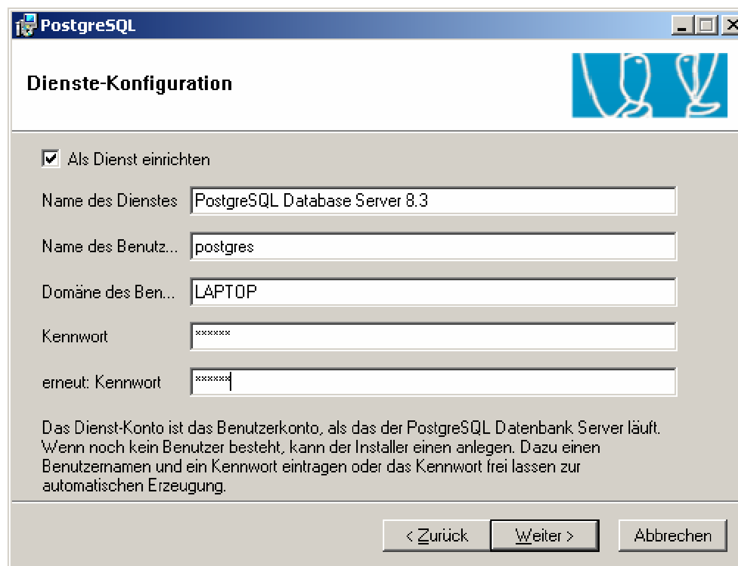
2. Installationsfile (postgresql-8.3.x msi) ausführen und Dialog folgen.

3. Sprache wählen

4. Installationsoptionen: Standardinstallation mit Benutzerschnittstellen „psql und pgAminIII". Weitere Optionen können auch nachträglich installiert werden.

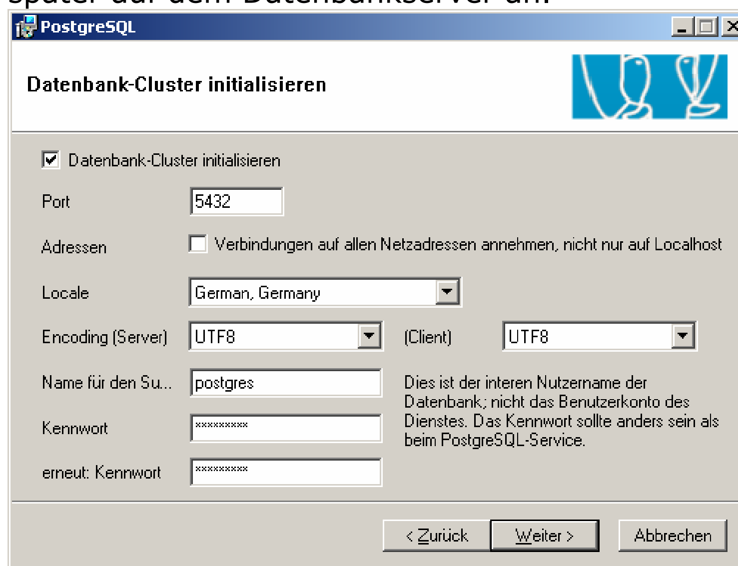


5. Dienste Konfiguration, als Dienst einrichten für Benutzer aus Schritt 1. Die Domäne des Benutzers wird in der Regel vom Installer automatisch eingestellt.



Logon as Service vom Installer einrichten lassen (Zwischendialog)

6. Datenbankcluster initialisieren:
   Vorschlag für Encoding: UTF8
   User postgres (oder ein anderer Name), Passwort vergeben (nicht das des Benutzers aus Schritt 1!). Mit diesem Passwort melden Sie sich später auf dem Datenbankserver an.



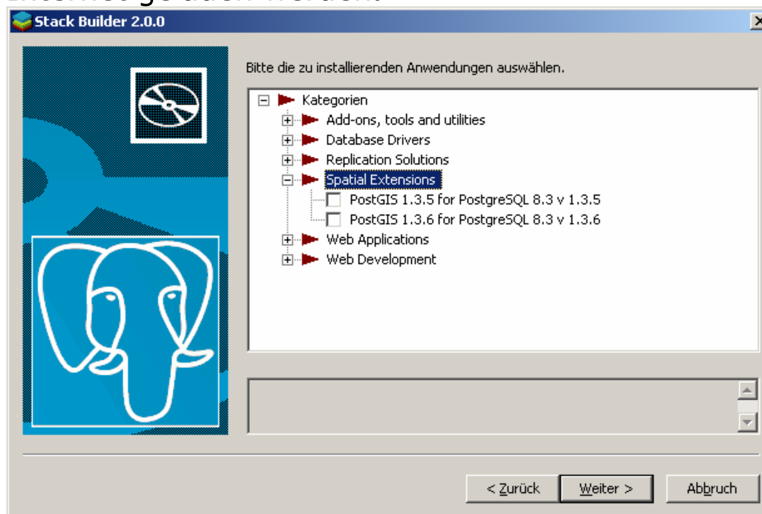
7. Dialog Prozedurale Sprachen: PL/pgsql aktivieren

8. Dialog Contrib-Module nur Administrator-Paket aktivieren

9. Installation für PostgreSQL ausführen und beenden.

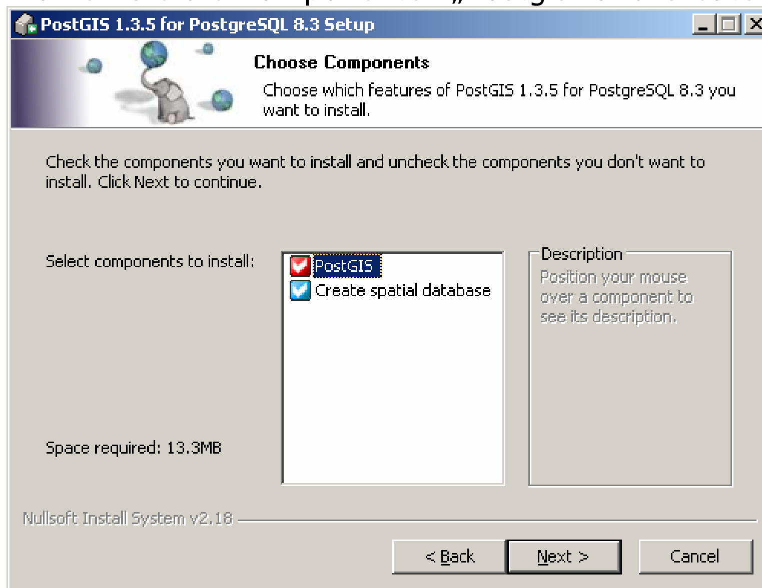
b) Installation Postgis

1. Bei weiterer Installation aus dem PostgreSQL-Dialog kann an dieser Stelle POSTGIS auch über die Option „Launch Stack Builder" über das Internet geladen werden.



   Sie können den Installer für Postgis auch downloaden und direkt ausführen (Empfehlung bei der Installation auf Servern, die aus Sicherheitsgründen den Download über das Internet nicht zulassen)

2. Wählen Sie die Komponenten „Postgis" und Create spatial database"



3. Im Dialog „choose install location" empfiehlt sich das Standardverzeichnis unter PostgreSQL

4. Nach dem Dialog „databaseconnection" bei dem Sie die Zugangsdaten des Datenbankservers eingeben und dem Dialog „databasename" bei dem die Standarddatenbank für Postgis geändert werden kann ist die Installation abgeschlossen.

c) Optimierung des Datenbankservers

Die Voreinstellungen einer Standardinstallation von PostgreSQL sind für die Anwendung in einem Produktivsystem nicht ratsam da die Einstellungen eher für den parallelen Betrieb in einer Entwicklungsumgebung oder für Systeme mit schwacher Hardwareausstattung konzipiert wurden. Es sollten deshalb einige Anpassungen in der Konfiguration vorgenommen werden.
In diesem Zusammenhang empfehlen wir die Lektüre "PostgreSQL Administration" von Peter Eisenkraut aus dem O'Reilly Verlag.
Alle Parameter zur Optimierung können mit einem Editor in der zentralen Konfigurationsdatei **postgresql.conf** vorgenommen werden. Bei einer Standardinstallation befindet sich unter Windows dem Verzeichnis *C:\Programme\PostgreSQL\8.3\data.* Die Beschreibung der optimalen Einstellungen der Parameter würde den Rahmen des Tutorials sprengen, es sollen deshalb lediglich die Parameter aufgelistet werden, die vor einer Inbetriebnahme angepasst oder überprüft werden sollten.

Einstellungen zur Verbindungskontrolle:
→ listen_adresses (vgl. Kap. Zugangskontrolle)
→ max_connections (maximale Anzahl von Verbindungen die eine Instanz des DB-Servers gleichzeitig offen haben kann)
→ ssl (aktiviert die SSL-Unterstützung des DB-Servers).

Einstellungen zur Speicherverwaltung:
→ shared_buffers (legt die Größe des Shared-Buffer-Pools einer DB-Instanz fest)
→ work_mem (Obergrenze des zur Verfügung stehenden Arbeitsspeichers für DB-Operationen wie Sortieren oder Verknüpfungsalgorithmen)
→ maintenance_work_mem (Obergrenze des zur Verfügung stehenden Arbeitsspeichers für Verwaltungsoperationen zur Erzeugung oder Änderung von DB-Objekten)
→ max_fsm_pages

Einstellungen zur Wartung, Planeinstellungen und Logging:
→ wal_buffers
→ checkpoint_segments
→ checkpoint_timeout
→ effective_cache_size
→ log_line_prefix

<u>Automatische Optimierung mit Zusatztool:</u>
Eine schnelle und gute Analyse des Servers gelingt mit dem kostenlos verfügbaren Tools **EDB Tuning Wizard** der Firma EnterpriseDB. Man kann es über den Application Stackbuilder in PostgreSQL installieren wenn der Server es zulässt. Es muss allerdings vorher einen Account bei Enterprisedb anlegt werden. Der Wizzard ist eigentlich selbsterklärend und erzeugt nach entsprechenden Voreinstellungen (Server-Utilization: Development, Mixed-Mode oder Dedicated) eine neue Konfigurationsdatei postgresql.conf. Die Änderungen werden dabei gekennzeichnet und die Originaldatei in einer Kopie gespeichert. Am Schluss muss lediglich der Datenbankdienst neu gestartet werden damit die neue Konfiguration wirksam wird.

2. Administration der Datenbank

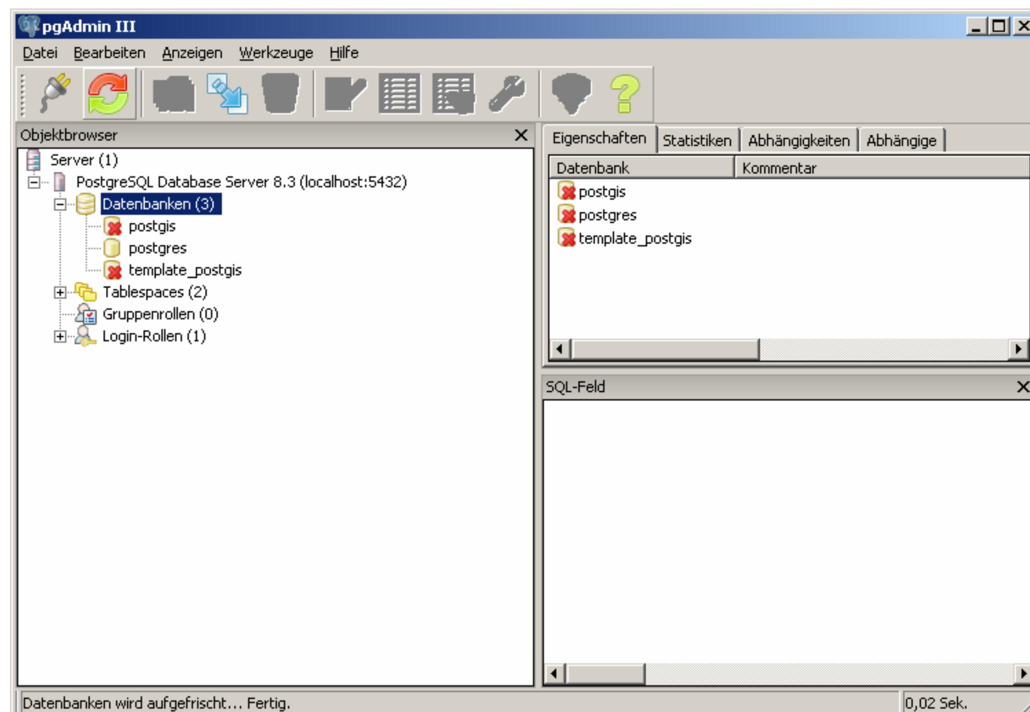
Zur Administration stehen nach der Standardinstallation unter Windows folgende Datenbank-Clients zur Verfügung:

Terminalprogramm psql

Für Freunde der Kommandozeile kann die Administration des Datenbankservers über den PostgreSQL-Terminal **psql** vorgenommen werden.
Sie öffnen das Kommandozeilenprogramm über
Start/Programme/PostgreSQL8.3/psql nach postgres
Sie sind direkt als Administrator „postgres" angemeldet und können sofort SQL-Statements z.B. zur Erzeugung einer Datenbank abschicken.
Über *Start/Programme/PostgreSQL8.3/Befehlszeile* können Sie direkt psql mit seiner Befehlsreferenz ausführen.
Sind Sie über psql auf einer Datenbank angemeldet muss jeder Befehl entweder mit /g oder einem Semikolon beendet werden.

pgAdmin III

Dieser Client ist eine graphische Benutzeroberfläche zur Administration von PostgreSQL. Sie Starten das Programm über
Start/Programme/PostgreSQL8.3/pgadmin III
Nach der Anmeldung sind die Datenbanken und deren Objekte in einer Baumstruktur im linken Fenster verfügbar. Es können alle Attribute und SQL-Befehle über diese Oberfläche komfortabel einstellen und abschicken.
Die folgende Referenz bezieht sich auf die Arbeit mit dem pgAdmin.
Alle Operationen können natürlich auch mit dem Terminalprogramm auf Komandozeilenebende vorgenommen werden.



3. Zugangskontrolle - Freigabe in einem Netzwerk

Nach der Installation ist die Datenbank lediglich vom lokalen Server erreichbar. Zur Freigabe der Datenbank in einem Netzwerk für andere Hosts müssen folgende Einstellungen vorgenommen werden (einfaches Beispiel):

postgresql.conf

Hier muß der Parameter „listen_addresses“ angepasst werden. Der Parameter konfiguriert die IP-Adressen auf denen Datenbankverbindungen entgegen genommen werden. Es können IP-Adressen oder Hostnamen eingestellt werden. Normalerweise wird die Voreinstellung so geändert, dass Verbindungen von allen Adressen (Platzhalter *) möglich sind:

listen_addresses = '*'

Andere Einstellungen siehe Dokumentation.

pg_hbd.conf

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
  host      all         all         192.168.178.8/32      md5
```

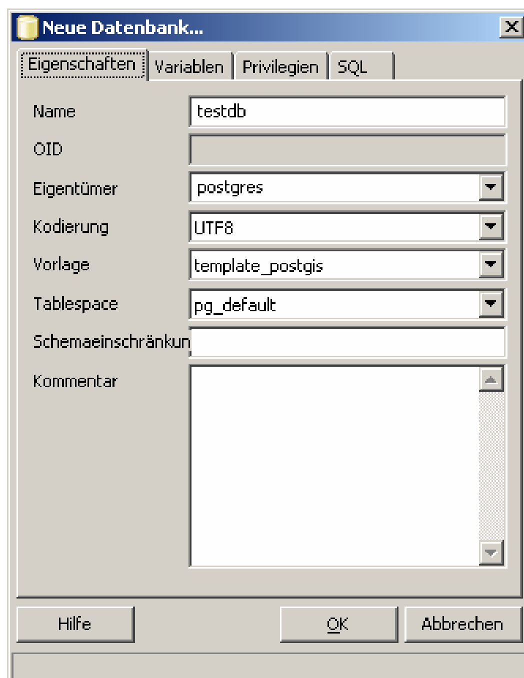
In diesem Fall erhält ein einzelner Host über Passwortabfrage die Berechtigung für alle Datenbanken und User, andere Einstellungen siehe Doku.

Bitte beachten Sie die entsprechende Portfreigabe in der Firewall des Servers!

4. Anlegen einer Datenbank

Bei der Installation von PostgreSQL wird automatisch eine Datenbank angelegt: *postgres*
Die Datenbank *postgres* ist die Standardvorlage zum Erstellen von neuen Datenbanken. Diese Datenbank sollte deshalb nicht gelöscht werden.

Nach der Installation von Postgis werden zwei zusätzliche Datenbanken angelegt: *postgis* und *template_postgis* (diese geht aus *postgis* hervor).
Zur Erstellung einer neuen „räumlichen" Datenbank sollte als Vorlage *template_postgis* verwendet werden. Zur Unterstützung aller Umlaute und Sprachen wird die Kodierung UTF8 empfohlen.



Die Datenbank template_postgis beinhaltet 2 Tabellen:

1. geometry_columns
   Diese Tabelle speichert die Informationen zu „räumlichen" Tabellen, wie den EPSG-Code (srid) und den Namen der Geometriespalte und Geometrietyp
2. spatial_ref_sys
   Die Tabelle speichert die Informationen zu den verschiedenen Projektionen. Diese Informationen werden z.B. auch für das Umwandeln von Daten in unterschiedliche Projektionssysteme verwendet.

5. Hilfsprogramme (Loader und Dumper)

Nach der Standardinstallation von PostGIS liegen im bin-Verzeichnis von PostgreSQL die zwei Kommandozeilenprogramme shp2pgsql.exe und pgsql2shp.exe mit denen es möglich wird PostGIS-Tabellen in das Shape-Format zu konvertieren und umgekehrt diese zu laden.
Tipp: Für eine komfortable Bearbeitung ohne Angabe von Programm-Pfaden empfehlen wir das Anlegen einer Systemvariablen auf das bin-Verzeichnis von PostgreSQL. Dazu gehen Sie über die Systemsteuerung auf System→ Erweitert → Umgebungsvariablen. Danach wählen Sie unter den Systemvariablen „Path" und fügen den Pfad zum bin-Verzeichnis (z.B. C:\Programme\PostgreSQL\8.3\bin) durch ein Semikolon getrennt hinzu.
Sie können nun die Kommandozeilenprogramme von jedem Ort auf dem Rechner ohne Angabe des Pfades aufrufen.

shp2pgsql
Mit diesem Programm können sehr komfortabel ESRI-Shape-Daten nach Postgis geladen werden.
Syntax:

Folgende Optionen stehen zur Verfügung:

-d Drops the database table before creating a new table with the data in the Shape file.
-a Appends data from the Shape file into the database table. Note that to use this option to load multiple files, the files must have the same attributes and same data types.
-c Creates a new table and populates it from the Shape file. This is the default mode.
-p Only produces the table creation SQL code, without adding any actual data. This can be used if you need to completely separate the table creation and data loading steps.
-D Use the PostgreSQL "dump" format for the output data. This can be combined with -a, -c and -d. It is much faster to load than the default "insert" SQL format. Use this for very large data sets.
-s <SRID> Creates and populates the geometry tables with the specified SRID.
-k Keep identifiers' case (column, schema and attributes). Note that attributes in Shapefile are all UPPERCASE.
-i Coerce all integers to standard 32-bit integers, do not create 64-bit bigints, even if the DBF header signature appears to warrant it.
-I Create a GiST index on the geometry column.
-w Output WKT format, for use with older (0.x) versions of PostGIS. Note that this will introduce coordinate drifts and will drop M values from shapefiles.
-W <encoding> Specify encoding of the input data (dbf file). When used, all attributes of the dbf are converted from the specified encoding to UTF8. The resulting SQL output will contain a SET CLIENT_ENCODING to UTF8 command, so that the backend will be able to reconvert from UTF8 to whatever encoding the database is configured to use internally.

Beispiele:
In diesem Beispiel wird die Datei "mytable" direkt in die Datenbank "mydb" durchgeladen. Dabei wird automatisch die Tabelle „mytable" angelegt und die entsprechenden Einträge in der Tabelle „geometry_columns" vorgenommen:

```
shp2pgsql -s 31466 -W Latin1 -I myshape  mytable | psql -U postgres -d mydb
```

Sehr komfortabel ist in diesem Zusammenhang auch das Anlegen von .bat-Dateien zum Laden von mehreren Shape-Dateien in einem Ordner, z.B.

```
@echo off
shp2pgsql -s 31466 -W Latin1 -I shape1 tabelle1 | psql -U postgres -d mydb
shp2pgsql -s 31466 -W Latin1 -I shape2 tabelle2 | psql -U postgres -d mydb
shp2pgsql -s 31466 -W Latin1 -I shape3 tabelle3 | psql -U postgres -d mydb
pause
```

Soll der File nicht direkt in die Datenbank durchgeladen werden kann auch ein SQL-File erzeugt werden, der später in der Datenbank ausgeführt wird:

```
shp2pgsql -s 31466 -W Latin1 - myshape  mytable > sqlshape.sql
```

pgsql2shp

Dieses Programm dient dem Export von PostgreSQL/PostGIS nach ESRI-Shape Hierbei können aus SQL-Statements abgesetzt werden.
Syntax:
pgsql2shp [<options>] <database> [<schema>.]<table>
pgsql2shp [<options>] <database> <query>

Folgende Optionen stehen zur Verfügung:
-f <filename> Write the output to a particular filename.
-h <host> The database host to connect to.
-p <port> The port to connect to on the database host.
-P <password> The password to use when connecting to the database.
-u <user> The username to use when connecting to the database.
-g <geometry column> In the case of tables with multiple geometry columns, the geometry column to use when writing the shape file.
-b Use a binary cursor. This will make the operation faster, but will not work if any NON-geometry attribute in the table lacks a cast to text.
-r Raw mode. Do not drop the gid field, or escape column names.
-d For backward compatibility: write a 3-dimensional shape file when dumping from old (pre-1.0.0) postgis databases (the default is to write a 2-dimensional shape file in that case). Starting from postgis-1.0.0+, dimensions are fully encoded.

Beispiele:
In diesem Beispiel wird die gesamte Tabelle „mytable‟ in den Shapefile shapename geschrieben:

```
pgsql2shp -f C:\meinverzeichnis\shapename mydb -h localhost -P xxxx -u postgres mytable
```

In diesem Beispiel werden über eine Boundingbox alle darin liegenden Objekte der Tabelle „mytable‟ ausgewählt:

```
pgsql2shp -f C:\meinverzeichnis\shapename -h localhost -P xxxx -u postgres mydb "SELECT * FROM mytable
WHERE the_geom && SetSRID('BOX3D(2618601.352688 5580443.902885,2634543.332106
5594406.159369)'::box3d,31466);"
```

6. Einbindung in UMN Mapserver

Postgis-Tabellen können direkt in UMN Mapserver eingebunden werden. Dabei wird im Layerobjekt der Connectentyp auf postgis gesetzt und das data-Objekt mit dem SQL-Statement belegt:

```
LAYER
  NAME "testlayer"
  TYPE polygon
  STATUS on
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=xxxx dbname=mydb
  host=localhost port=5432"
  DATA "the_geom from mytable  using unique gid"
  CLASS
      NAME "test"
      OUTLINECOLOR 200 0 200
    END
END
```

Es Können auch komplexere Statement abgesetzt werden. Im folgenden Beispiel werden die Mittelpunktkoordinaten aus einem Polygontabelle abgefragt und diese als Punktthema dargestellt:

```
LAYER
  NAME "testlayer "
  TYPE point
  STATUS on
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=xxxx dbname=mydb host=localhost port=5432"
  DATA "the_geom FROM (SELECT gid, GeometryFromText('POINT(' || x(Centroid(the_geom)) || ' ' ||
   y(Centroid(the_geom)) || ')', 31466) AS the_geom FROM mytable) as foo using unique gid using
   SRID=31466' "
  TEMPLATE "template/query.html"
    CLASS
     NAME "test"
     SYMBOL "testsymbol"
     SIZE 20
     END
END
```

7. Räumliche Operationen mit Postgis

Basiswissen SQL:
Ein Grundwissen bzgl. SQL (Structured Query Language) als Standardsprache für den Zugriff auf Datenbanken wird an dieser Stelle vorausgesetzt.

Basiswissen zu PostgreSQL und PostGIS:

Groß- und Kleinschreibung:
PostgreSQL interpretiert alle Schlüsselwörter und Bezeichner als Kleinbuchstaben. Deshalb sollten Tabellen- und Feldnamen in Kleinbuchstaben gehalten werden, um Fehler zu vermeiden.

Wichtige Datentypen:
In der Regel kommen nur recht wenige der zahlreichen verfügbaren Datentypen zum Einsatz. Hier eine Auswahl der meist gebräuchlichen Datentypen:

int4, int8 → ganze Zahlen (4byte/8byte)
float8 → Dezimalzahlen (double precision)
varchar → Textfelder mit variabler Zeichenlänge
text → Memofeld, Textfeld mit unbegrenzter Länge
bool → Boolean (true/false)
date → Datumsangebe im Standardformat YYYY-MM-DD, z.B. 2009-10-01

OGC Konform
Die Datenhaltung einer räumlichen Tabelle in PostgreSQL mit PostGIS folgt den *OGC Simple Feature Specifications für SQL*. Die OGC Spezifikation definiert ein SQLSchema zur Verwaltung von Geodaten.
Eine Feature Table enthält in der Regel einen Geometrietyp (Punkt, Linie, Polygon). Ausnahme: Geometrycollection. Die Geometrieobjekte werden dabei in eigenem Tabellenfeld gespeichert. Das Format der Geometrieobjekte ist das WKT-Format (Well Known Text), intern werden die Geometrien im WKB-Format (Well Known Binary) gehalten.
Hier eine Übersicht der Geometrietypen im WKT-Format (Quelle: OpenGIS Simple Features Specification for SQL):

Geometry Type	SQL Text Literal Representation	Comment
Point	'POINT (10 10)'	a Point
LineString	'LINESTRING (10 10, 20 20, 30 40)'	a LineString with 3 points
Polygon	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'	a Polygon with 1 exterior ring and 0 interior rings
Multipoint	'MULTIPOINT (10 10, 20 20)'	a MultiPoint with 2 point
MultiLineString	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'	a MultiLineString with 2 linestrings
MultiPolygon	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'	a MultiPolygon with 2 polygons
GeomCollection	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'	a GeometryCollection consisting of 2 Point values and a LineString value

Erzeugen von Geodaten via SQL:

Beispiel Punktthema:

→ Anlegen einer Tabelle

```
CREATE TABLE punkttabelle (gid int4, name  varchar);
```

→ Einfügen der Geometriespalte und Eintrag in Metadatentabelle über die Funktion **AddGeometryColumn()**

```
SELECT AddGeometryColumn('public','punkttabelle','the_geom',31466,'POINT',2);
```

→ Einfügen von Daten mit Hilfe der Funktion **GeometryFromText()**
Die Geometriedaten werden hier in Form des WKT-Formats eingefügt.

```
INSERT INTO punkttabelle values ('1','Name1',GeometryFromText('POINT(2545882 5513130)', 31466));
INSERT INTO punkttabelle values ('2', 'Name2',GeometryFromText('POINT(2546495 5513862)', 31466));
INSERT INTO punkttabelle values ('3', 'Name2',GeometryFromText('POINT(2546400 5513890)', 31466));
```

Die Daten sind jetzt eingefügt und können bereits mit einem Client dargestellt werden.

→ Bei der Sichtung der Tabelle im pgadmin kommt der berechtigte Hinweis, dass kein Primärschlüssel vorhanden ist, deshalb hier noch das entsprechende SQL:

```
ALTER TABLE punkttabelle ADD CONSTRAINT punkttabelle_pkey PRIMARY KEY(gid);
```

→ Gerade bei großen Tabellen sollte aus Performancegründen ein räumlicher Index auf die Geometriespalte gelegt werden.

```
CREATE INDEX punkttabelle_the_geom_gist ON punkttabelle  USING gist (the_geom);
```

Hier noch einmal die komplette Syntax zum Erzeugen einer räumlichen Punkttabelle mit Bordmittel von POSTGIS:

```
CREATE TABLE punkttabelle (gid int4, name  varchar);
SELECT AddGeometryColumn('public','punkttabelle','the_geom',31466,'POINT',2);
ALTER TABLE punkttabelle ADD CONSTRAINT punkttabelle_pkey PRIMARY KEY(gid);
INSERT INTO punkttabelle values ('1','Name1',GeometryFromText('POINT(2545882 5513130)', 31466));
INSERT INTO punkttabelle values ('2', 'Name2',GeometryFromText('POINT(2546495 5513862)', 31466));
INSERT INTO punkttabelle values ('3', 'Name2',GeometryFromText('POINT(2546400 5513890)', 31466));
CREATE INDEX punkttabelle_the_geom_gist ON punkttabelle  USING gist (the_geom);
```

Das Erzeugen von Linien bzw. Polygonthemen funktioniert analog unter Verwendung der entsprechenden WKT Syntax, z.B.

```
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
```

Besser ist die direkte Verwendung von Multiobjekttypen

```
MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
```

Sauberes Löschen einer Geometrietabelle:

→ Zum Löschen der Geometriespalte und des Eintrags in der Metadatentabelle sollte folgende Funktion Verwendung finden: **DropGeometryColumn()**

```
Select DropGeometryTable ('punkttabelle');
```

Fehler in Geometrien aufspüren:

→ Mit der Funktion **ST_IsValid()** können Sie überprüfen, ob eine Geometrie "Wohlgeformt" ist also eine OGC-Konforme Geometrie ist:

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As konform,
        ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As defekt
```

Ausgabe von Geometriedaten im WKT-Format:

→ Um Daten einer bestehenden Tabelle wieder in einem lesbaren WKT-Format auszugeben dient die Funktion **ST_AsEWKT()**:

```
Select ST_AsEWKT(the_geom) from punkttabelle;
```

Ausgegeben wird der EPSG-Code in Verbindung mit dem WKT-String:

```
> SRID=31466;POINT(2545882 5513130)
> SRID=31466;POINT(2546495 5513862)
> SRID=31466;POINT(2546400 5513890)
```

Ausgabe der räumlichen Ausdehnung:

→ Die räumliche Ausdehnung lässt sich mit der Funktion **ST_EXTENT()** ausgeben. Dabei wird eine Bounding-Box der begrenzenden Koordinaten zurückgegeben (südwestlichster Punkt / nordöstlichster Punkt):

```
SELECT ST_EXTENT(the_geom) FROM punkttabelle;
```

Ausgabe:

```
> BOX(2545882 5513130,2546495 5513890)
```

Ausgabe von Punktkoordinaten:

→ Um an die X- bzw Y Koordinaten (double) eines Punktthemas direkt auszugeben kommt die Funktion **ST_X()** bzw. **ST_Y()** zum Einsatz:

```
SELECT ST_X(the_geom), ST_Y(the_geom) FROM punkttabelle;
```

→ Um z.B. an die jeweiligen Anfangspunktkoordinaten eines Linienobjektes zu gelangen, kommt eine Kombination aus ST_X bzw. ST_Y und **ST_POINTN** zum Einsatz:

```
SELECT st_x(st_pointn(the_geom,1)), st_y(st_pointn(the_geom,1)) FROM linientabelle;
```

→ Um den Centroid eines Polygons, Multipoints oder Linestrings auszugeben verwenden Sie **ST_Centroid()** (in diesem Beispiel in Verbindung mit der Ausgabe als WKT):

```
SELECT ST_AsText(ST_Centroid(the_geom)) FROM polygontabelle
```

Ausgabe von Flächen und Längen sowie Umfang:

→ Die Funktion **ST_AREA()** gibt die Fläche eines Polygons als double-Wert zurück (an dieser Stelle ein kleiner Ausflug, **ilike** hilft Ihnen bei PostgreSQL einen Textstring zu finden ohne auf Groß oder Kleinschreibung zu achten):

```
SELECT ST_AREA(the_geom) FROM polygontabelle WHERE name ilike 'name1';
```

→ Die Funktion **ST_LENGTH()** gibt die jeweilige Länge eines Linienobjektes zurück

```
SELECT ST_LENGTH(the_geom) FROM linientabelle;
```

→ Die Funktion **ST_PERIMETER()** gibt den jeweiligen Umfang eines Polygonobjektes zurück

```
SELECT ST_PERIMETER(the_geom) FROM polygontabelle;
```

Geometrische Operationen

→ Einen Buffer von z.B. 100 Metern auf alle Objekte eines Linienthemas erzeugen wir mit der Funktion **st_buffer():**

```
SELECT st_buffer(the_geom,100) as buffer_100 FROM linientabelle;
```

→ Um diese Thema auch in einem Client (z.B. QGIS) als hinzufügbares Thema verwenden zu können und nur auf ein Objekt zu beschränken (Name1), bilden wir einen View auf das Bufferthema. Die Funktion **ST_SetSRID** dient in diesem Zusammenhang einer Zuordnung zu einem EPSG-Code.

```
CREATE VIEW buffer_linie_100 AS
 SELECT gid, name, ST_SetSRID(st_buffer(the_geom,100), 31466 ) AS
 the_geom  FROM linientabelle WHERE name = 'Name1';
```

→ Die Funktion **ST_Intersection** ermöglicht klassische Verschneidungsoperationen in PostGIS, zurück bekommt man die

Verschneidungsgeometrie für die Objekte die nicht leer sind (**IsEmpty** = FALSE):

```
SELECT p.gid, p.name, ST_SetSRID(ST_Intersection(p.the_geom, b.the_geom) ,31466)
 AS the_geom  FROM polygontabelle AS p, buffer_linie_100 AS b
WHERE IsEmpty (ST_Intersection(p.the_geom, b.the_geom) ) = FALSE;
```

→ Um dieses Thema wiederum in einem Client sichtbar zu machen erzeugen wir einen View:

```
CREATE VIEW intersect_polygon (gid,name,the_geom) AS
SELECT p.gid, p.name, ST_SetSRID(ST_Intersection(p.the_geom, b.the_geom) ,31466)
 AS the_geom  FROM polygontabelle AS p, buffer_linie_100 AS b
WHERE IsEmpty (ST_Intersection(p.the_geom, b.the_geom) ) = FALSE;
```

→ Um eine konvexe Hülle um eine Punktwolke zu erzeugen benötigen wir die Funktion **ST_ConvexHull** und **ST_Union**

```
SELECT ST_SetSRID(ST_ConvexHull(ST_Union(the_geom )),31466) as the_geom FROM punkttabelle;
```

→ Um festzustellen ob ein Polygon einen Punkt enthält, kann die Funktion **ST_Contains** zur Anwendung kommen:

```
SELECT po.name AS Polygon_name, pu.name AS Punkt_name
 FROM polygontabelle AS po, punkttabelle AS pu
 WHERE ST_Contains (po.the_geom, pu.the_geom ) = TRUE;
```

→ Ein weiterer Lösungsweg ist die Funktion **ST_Within**, hier wird allerdings ermittelt, ob die Punktgeometrien in den Polygonen enthalten sind:

```
SELECT po.name AS Polygon_name, pu.name AS Punkt_name
 FROM polygontabelle AS po, punkttabelle AS pu
 WHERE ST_Within (pu.the_geom, po.the_geom ) = TRUE;
```

Weitere Beispiele aus der Praxis:

Attribute eines aus einem Thema (usergrenzen_sgdn) in ein Feld eines anderen Themas (biogas_sgdn.edituser) übernehmen, wenn Objekt darin liegt:

```
UPDATE biogas_sgdn SET edituser = u.edituser FROM usergrenzen_sgdn AS u WHERE CONTAINS(u.the_geom, biogas_sgdn.the_geom);
```

Linienobjekt aus WKT einfügen:

```
INSERT INTO "messprofile"
("projekt","profil","datum","edituser",the_geom) VALUES
('Baldeneysee-
Elodea','Profil01','01.09.2006','edit',GeomFromEWKT('SRID=31467;MULTILINESTRING((3366132.18
5696901.85,3365966.09 5696925.09))'));
```

Geometrietyp ermitteln:

```
select GeometryType(the_geom) from siedlungsflaechen_fnp;
```

Stringverknüpfung herstellen:

```
UPDATE grenzen_gemeinden SET gem_schl= '07' || gemeinde_n;
```

Substrings ermitteln und verknüpfen

```
update gem_fl set gem_schl_neu = SUBSTR(gem_schl,0,6)||'01'||SUBSTR(gem_schl,6,4);
```

Feld hinzufügen und mit festem Wert besetzen:

```
ALTER TABLE immobilien ADD COLUMN edituser character varying(50);
update immobilien set edituser = 'administrator';
```

Feld in zu einen anderen Typ casten:

```
update gemarkungen set gem_nr_new = cast(gem_nr as integer);
```

Punktview aus zwei Koordinaten einer nichträumlichen Tabelle erzeugen:

```
CREATE VIEW geopoint AS
SELECT lfd_nr, map_text,
ST_SetSRID(GeometryFromText('POINT(' || r_gk2 ||' '|| h_gk2 || ')', 31466), 31466 ) AS the_geom
FROM kultur;

CREATE VIEW geopoint_wgs AS
SELECT lfd_nr, map_text,
ST_SetSRID(GeometryFromText('POINT(' || x_wgs84 ||' '|| y_wgs84 || ')', 4326), 4326 ) AS the_geom
FROM kultur;
```

Ersetzen von Kommas durch Punkte in String und gleichzeitiges Casten:

```
update kultur set x_wgs84 = cast(REPLACE(r_wgs84,',','.') as double precision);
```